# PASSMADE Principle for Software Architecture

The following information is being provided as a convenience to California Business Connect bidders from an Allied Consultants blog located at **http://www.alliedc.com/blog/?p=501** published April 21st, 2011 by Ammar Ahmed Khan:

## PASSMADE Principle for Software Architecture

Recently I've come across an acronym called PASSMADE, which is interesting to memorize some important principles on Software Development; it looks like this:

**P**erformance
**A**vailability
**S**ecurity
**S**calability
**M**aintainability
**A**ccessibility
**D**eployability
**E**xtensibility

We'll have a brief look at each of these and then look at a few other possibilities for requirements. Remember, though, that all of these potential requirements should be evaluated in a cost vs. potential benefits manner before including them in the specifications.

**Performance** requirements usually deal with a system's response time, which is defined as the amount of time a user waits to receive a response after issuing some action. This is usually a more significant set of requirements on Web sites and enterprise applications than on Windows-desktop applications. Some of the major factors that affect response time include the following:

- Limited Bandwidth
- Server or network capacity
- Server traffic
- Application-processing constraints

Some of these factors are outside of your control, so the point is to reduce unnecessary overhead and performance bottlenecks as much as possible. You may also want to gauge how much time the application takes to respond using some automated tools. However, be careful not to mix up performance requirements with other operational requirements, such as availability and scalability.

**Availability** requirements deal with how often your application is online to do its specified work. In looking at availability, there is a bit of overlap with "Reliability" as well. Reliability is the capability of a component or service to perform its functions at a specific time or over a period of time. Essentially, availability = reliability + restore time (time required to bring a system back to normal after a failure).

In deciding whether to include availability requirements, evaluate how often customers need it online. What are the peak hours? Is time used for system backups acceptable? What is the schedule and budget? Remember, before you include a requirement, make sure it's needed by the customer/client.

Sometimes, when availability requirements are not addressed, some of the following issues can result in a problematic solution for the end user:

- Inadequate testing
- Change management problems
- Lack of ongoing monitoring and analysis
- Operational errors
- Weak program code
- Interactions with external services or applications
- Different operating conditions (usage level changes, peak overloads)
- Hardware failures (disks, controllers, network devices, servers, power supplies, memory, CPU)
- Natural disasters

Obviously, you can't account for all this, because some items are not under your control. However, identifying availability requirements from the get-go can help you avoid some of these problems down the road.

**Security** requirements are probably the most important requirements you'll come across when developing a solution. The most important facet of developing these requirements is to understand what your application needs protection from. As a reminder, most security solutions are based on the following principles:

- Authentication – The process of identifying who someone is, using some form of credentials. In the .NET world, this is usually through Windows, Forms, Passport, or Custom authentication.
- Authorization – The process of identifying what resources someone has access to after being authenticated.
- Data Protection – The process of providing data confidentiality and integrity. Data needs to be secure both while in transit and while in storage. One method to accomplish this goal is to encrypt the data.
- Auditing – The process of monitoring and logging what goes on in a system that relates to security.

Some example requirements could include things like encouragement of strong password use, logon policies and auditing (passwords expire after a certain amount of time), intruder prevention processes like using a security question and email to offer change password services, ownership/responsibility for user accounts (suspending accounts that have abused the system), and encryption policies for data protection.

One acronym that Microsoft has coined for security is STRIDE, which stands for the following:

- **S**poofing Identity – Includes anything done to compromise a trusted user's authentication information, such as a username and password.
- **T**ampering with Data – Includes anything from ill-mannered modification of database data to defacement of the user interface. Another topic in this category is session hijacking.
- **R**epudiation – Involves users that deny that they have performed a certain action, and the other parties have no way of proving either case. This usually results from lax logging and auditing policies.
- **I**nformation Disclosure – Includes times when confidential information is exposed to individuals who should not have access to that information.
- **D**enial of Service – Includes directed attacks against a host or network to disallow other legitimate users from accessing the site. These kinds of attacks usually overload the host with too much traffic and can include multiple simultaneous attackers.
- **E**levation of Privilege – Includes giving an unprivileged user privileges to compromise or destroy system data. Authorization is a very important mechanism for enforcing this.

For more thorough information about STRIDE, look here[1].

Finally, consider the business's perspective in this. How sensitive is the data that the application will store? What is the effect if the data gets out to the wrong people? Also, consider how security policies can be improved even after deployment. You may want to consider a periodic security review that evaluates the risk of new threats on the system. Organizations will really have to remain vigilant in order to protect their data.

**Scalability** requirements involve the capability of a system to increase its capacity by upgrading system hardware and software.

Scalability deals mostly with distributed/enterprise applications—standalone applications will not usually need to achieve much when it comes to scalability. Although implementation of these requirements come during the physical design and development phases, the requirements should be considered early because you need to be aware of what changes should be made to the current system's hardware and/or that the software should be designed to fit for an increase of ____ (amount) over ____ (time). Also, keep in mind that trying to maximize performance may also contradict scalability; the two are not the same.

There are two buzzwords that you may've heard of before: *scaling up* and *scaling out*. Scaling up is the process of achieving scalability through the use of faster/better (and usually more expensive) hardware. One thing to keep in mind while scaling up is that there are limitations to how much you can improve. Also, it is problematic to spend money on having one computer support the operation because it enables the application to have a single point of failure.

Scaling out is the process of using many machines to work as one machine by increasing scalability through hardware improvements, decreasing dependency on one machine, and allows an application to stay up even if a server fails. When scaling out there are things to keep in mind, such as load-balancing and state management (for web sites). For example, you can't use the HttpSessionState object (Session property of the Page class) to store state if you're using multiple machines. Instead, you would have to use ASP.NET's Server State or a DBMS (most likely SQL Server) to manage it. *The application logic should not have to change depending on which server the code is running on*. This is known as location transparency.

Next week we'll look at the remaining four categories of requirements as well as some additional categories that deal with user and business requirements, not just operational requirements.

**Maintainability** requirements describe how easily a software system can be modified to correct bugs, add new functionality desired by a client, or to improve performance. In a nutshell, it is a measure of how easy it is to keep the solution functioning correctly. The requirements are usually characterized with some sort of number to describe how often or to what extent the system will be serviced or unavailable. Here are some examples of maintainability requirements:

- Maintenance work on the system will be limited to *ten* days throughout the year, and each of those work days will not exceed *eight* hours.
- Minor functionality updates to the system will take no more than *five* days.
- Any period of unexpected downtime will not exceed *twenty* minutes.

When looking at these requirements, it outlines the main objectives of maintainability requirements:

[1] http://technet.microsoft.com/library/cc163159

- Minimize maintenance costs and allow for programmers to work on other projects instead of doing maintenance work.
- Ensure that minor updates to the application don't consume your resources.
- Ensure that defects and errors in your application don't take long to correct.

Areas of concern with maintainability requirements usually revolve around the following areas:

- *Breadth of the Application* – Across how many locations will the solution be installed? If it's installed in only one location, it (obviously) makes maintenance work a lot easier than it would be with a multi-site installation.
- *Method of Distribution* - If you are installing the solution across multiple sites, how are you going to make updates to your software universal? In essence, *how* are you going to keep your software properly maintained?
- *Maintenance Expectations* – It is important to discuss the maintenance work the client expects and come to a solution before writing anything in a formal document.
- *Impact of Third-Party Agreements* – If you are using third-party components to develop your application, you need to put a process in place to account for updates and bug fixes.

To summarize, maintainability requirements should focus on quantitative answers to problems such as modifications to improve functionality, corrections of minor defects (along with documentation updates). The areas these problems should revolve around include the time and expense needed to repair defects, account for system downtime, and staff locations with support teams.

**Accessibility** requirements focus on making your software as flexible as possible to accommodate for a wide range of users' needs with regard to disabilities and impedances that would make their experience of your software less fruitful.

Before you get started with accessibility requirements, it is important to consider your user base. Talk with your client about how many of the potential users of this software will need assistive support from the software. Usually the potential users of your software are asked to fill out a *needs assessment* document, which outlines accessibility requirements from the employees' perspective, the organization's processes, and the current technology. From the results of this *needs assessment*, you and the client can start to think about purchase decisions to take accessibility needs into consideration. The chances are that you will need to incorporate accessibility into your final solution, so the rest of this section assumes that.

For some users, accessibility could simply mean being able to change font size, colors, sounds, and other display elements. For others with more serious impairments, *assistive technology products* are used to accommodate appropriately. Remember that a lot of these products are already built into the operating system, like Microsoft Narrator. Here are some areas in which to consider implementing accessibility:

- In areas that affect a large number of users.
- In areas that your users will use frequently.
- In areas that are integral and necessary for your application to work according to specification.
- Application-wide such that it is compatible with accessibility aids.

The final topic in this section looks at web-based solutions. It is estimated that about 8% of all web users are disabled, so making a website that is accessibility-enabled can reach many more users than a non-accessibility-enabled one. Consider users with slow connections or text-only browsers when designing your application.

**Deployment** requirements focus on your users. Who are they and how are they getting your application? How many sites are you deploying to? Which users require solutions in different languages?

Users can be broken down into three categories:

- *Standalone Users* – Users who are not connected to a LAN.
- *Remote Access Users* – Users who will gain access to the network through remote access modems.
- *Local Network Users* **–** Users who are directly connected to the network.

The important thing to remember is that you need to understand the needs of each of these users to have a successful deployment. Collect information from your project teams, staff, and user groups. Surveys and interviews are also good supplements to figuring out how best to deploy a solution.

Remember that even if an application runs correctly in the development environment that there is a still a chance for it to crash and burn in the production environment. End users might be missing critical components that the project team assumed was on all computers; the components may even be outdated as well. One way to avoid this is assume that your target computers don't have any critical components on there, meaning that you have to install them with your software. But this doesn't stop the DLL Hell "phenomenon" most of us are familiar with, as all of those components could later become corrupted.

The last thing to mention is about distributing your application. How will it make its way to your end users? This will mostly depend on the location and number of your users as well as what category they fit into (see above). Here are some common deployment scenarios:

- For a large number of users in diverse and remote locations you might want to consider deploying through CD-ROM.
- For a smaller number of users you should consider deploying through the Internet or over an intranet.
- For users that are located worldwide you should consider deploying over the Internet with different packages for different language accommodations.

**Extensibility** requirements look at the adaptability of a system and the degree to which that system can be enhanced in the future. There are two main tenets of extensibility that come up every time a new feature is requested for an application: cost and time.

When designing your application, you should think about the direction in which the application is likely to go after the initial deployment phase. Of course it is good to talk to stakeholders, your client, and members of your project group to create a clearer (though it would be impossible to make it crystal-clear) vision of where the project is going. Systems have to remain flexible to account for the changing nature of business, because if those systems become out-of-date, then it will be trashed. This is a constant problem that software developers have to face, and considering extensibility is a good first step toward alleviating the problem.

On a more concrete note, when developing extensibility requirements, think about the average time and the cost it will take to make updates in different areas of the application. After making these requirements, design defensively! It will save you a lot of time in the long run when the client requests an update to the application.

In looking at PASSMADE there are a slew of areas that you have may not have considered when designing your application, but these are just operational requirements. Other requirements exist like business requirements (how your application fits with the business's organizational structure,

performing gap analysis, and modeling data flow) and user requirements (use cases, globalization, and localization concerns) which should be performed before you even make it to the PASSMADE level. Also, you will need to use your analysis from PASSMADE to determine whether the production environment has the right hardware and infrastructure to support your application. If not, you and your client will need to reach an agreement on how their IT infrastructure will support the current application and *any future releases of your application*. A substantial amount of planning and communication should occur to reach the right conclusion on these issues, and they will surely save time later down the road.